UNA VISIÓN DEL DESARROLLO DE SOFTWARE UTILIZANDO MODELOS

A VIEW OF SOFTWARE DEVELOMENT USING MODELS

AUTOR
LUIS OLIVERIO CHAPARRO LEMUS
Doctor (c) en Programación Declarativa e Ingeniería de la Programación
*Universidad Politécnica de Valencia
Docente Investigador
Programa de Ingeniería de Sistemas
lochaparro@uniboyaca.edu.co
COLOMBIA

AUTOR
JUAN FEDERICO GÓMEZ ESTUPIÑAN
Magister (c) en Ciencias de la Información y las Comunicaciones
**Universidad Distrital Francisco José de Caldas
Docente Investigador
Programa de Ingeniería de Sistemas
jfgomez@uniboyaca.edu.co
COLOMBIA

INSTITUCIÓN
*UNIVERSIDAD POLITÉCNICA DE VALENCIA
UPV
Camino de Vera, s/n 46022 Valencia
informacion@upv.es
ESPAÑA

INSTITUCIÓN
**UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
UDISTRITAL
Carrera 7 No. 40B - 53 admisiones@udistrital.edu.co
COLOMBIA

RECEPCIÓN: Febrero 29 de 2012 **ACEPTACIÓN:** Marzo 10 de 2012

TEMÁTICA: Nuevas Herramientas y Tecnologías de Desarrollo de Software.

TIPO DE ARTÍCULO: Artículo de Revisión

RESUMEN ANALÍTICO

La construcción de software está lejos de ser una tarea sencilla, sobre todo si su complejidad es alta. Esta tarea exige un alto compromiso del equipo de desarrollo, recursos costosos, especialistas altamente cualificados y procesos y métodos cada vez más formales. Con el propósito de agilizar este proceso, surgió un movimiento centrado en el uso de modelos en diferentes niveles de abstracción. Las principales propuestas en este sentido son la Arquitectura Dirigida por Modelos y el Desarrollo de Software Dirigido por Modelos (MDA y MDSD, por sus siglas en ingles). En este artículo se hace una revisión de la literatura a cerca de estas dos propuestas. Describe sus principios fundamentales, el trabajo orientado a conformar la fundamentación teórica de las dos propuestas y las principales herramientas que implementan el desarrollo conducido por modelos. Este trabajo se realiza en el marco del proyecto de investigación "Construcción de un proceso de desarrollo de software con base en MDA y MDSD", concretamente contribuye a establecer el trabajo de investigación realizado por la comunidad internacional y los fundamentos teóricos y conceptuales que subyacen a las dos propuestas.

PALABRAS CLAVES: Arquitectura Guiada por Modelos, Desarrollo de Software Guiado por Modelos, Modelo Independiente de Plataforma, Modelo Independiente de la Computación, Transformaciones entre modelos, Herramientas MDSD

ANALYTICAL SUMMARY

Building software is not an easy task, especially if its complexity is high. This task requires a high commitment to the development team, expensive resources, highly qualified specialists and increasingly formal methods and processes. In order to expedite this process, a movement focused on the use of models in different abstraction levels has emerged. The main proposals in this regard are Model Driven Architecture (MDA) and Model Driven Software Development (MDSD). This paper is a review of the literature about those two proposals. It describes its fundamental principles, work oriented to form its theoretical basis and the main tools that implement the model-driven development. This work is performed under the research project "Construction of a software development process based on MDA and MDSD", specifically contributes to establishing the research work done by the international community and the theoretical and conceptual underpinning the two proposals.

KEYWORDS: Model Driven Architecture MDA, Model Driven Software Development MDSD, Platform Independent Model PIM, Computation Independent Model CIM, Transformations between Models MDSD tools

INTRODUCCIÓN

La historia de la computación ha tenido una invariante: "el acercamiento progresivo del lenguaje de computación al lenguaje los usuarios humanos". Uno de los objetivos actuales de la investigación en computación es avanzar un paso más en este sentido. De los lenguajes de tercera generación con ayudas gráficas y asistencia CASE para actividades puntuales (como edición y compilación) se quiere saltar a los lenguajes fundamentados en la utilización de modelos. En este trabajo se realiza una revisión de la literatura acerca del desarrollo de software conducido por modelos.

El desarrollo conducido por modelos pretende liberar al ingeniero de los aspectos tecnológicos (programación) para centrarlo en las actividades de ingeniería (arquitectura, modelado y diseño). De manera similar a

como COBOL liberó al programador de las minucias del Hardware [1]

Los primeros intentos de usar los modelos en el proceso de desarrollo, empezaron con los diagramas propuestos en el análisis y diseño estructurado [2] Esta práctica evolucionó con el surgimiento del Análisis y Diseño Orientado a Objetos y la posterior introducción del Lenguaje Unificado de Modelado (UML por sus siglas en inglés) [3], [4] Sin embargo, las primeras prácticas de modelado estaban orientadas prioritariamente a documentar el sistema. La idea del desarrollo de software guiado por modelos, es un verdadero hito, en la historia del desarrollo de software, ya que pretende incorporar los modelos como parte viva de todo el proceso de desarrollo.

Los principales promotores de esta idea son el Grupo de Gestión de Objetos (OMG por sus sigla en inglés) [5] y un movimiento mundial de investigadores teóricos y prácticos agrupada en lo que se denomina "Desarrollo de Software Guiado por Modelos" MDSD, también conocida como "Desarrollo Guiado por Modelos" MDD [6], [7] El principio fundamental de estas dos corrientes es la utilización de modelos para desarrollar software, y su principal objetivo es la construcción de herramientas prácticas y los conceptos que conduzcan a la automatización del proceso de software.

Por su parte OMG, ha desarrollado un conjunto de estándares, en evolución permanente, denominada "Arquitectura Conducida por Modelos" MDA [8] Los principios fundamentales de MDA para la construcción de software son: utilización de modelos en todo el proceso de desarrollo, definición de un modelo para cada nivel de abstracción y transformaciones entre modelos. Estos principios favorecen la automatización del proceso de desarrollo de software de manera integral, desde sus etapas iniciales hasta la obtención de código ejecutable.

Para implementar el principio de niveles de abstracción, MDA ha definido tres modelos para describir el sistema [8]: el Modelo Independiente de la Computación CIM, el Modelo independiente de la Plataforma PIM y el Modelo Específico de la Plataforma PSM. El nivel de mayor abstracción es el CIM y representa el modelo del negocio desde el punto de vista del usuario. El PIM es un modelo conceptual que representa el punto de vista del usuario interpretado por el ingeniero de software. El PSM es un modelo del sistema escrito con el propósito de trasladarlo a una plataforma tecnológica específica.

Para presentar una visión significativa y coherente de las propuestas antes enunciadas, en este artículo se desarrolla el siguiente contenido: en el segundo capítulo se presenta la evolución de la computación desde el punto de vista de los lenguajes de programación; el tercer capítulo describe las similitudes y diferencias entre MDA y MDSD; el cuarto capítulo describe los fundamentos teóricos de las dos propuestas; el quinto capítulo presenta principales herramientas practicas y conceptuales que soportan y/o son soportadas por el desarrollo guiado por modelos; en el capítulo sexto se enuncian las tendencias hacia las que apunta la evolución del desarrollo guiado por modelos; y en el séptimo capítulo se presentan las trabajos futuros.

1. EVOLUCIÓN DEL FOCO DE LA COMPUTACIÓN: LENGUAJES DE PROGRAMACIÓN

La computación ha evolucionado desde la programación binaria a los lenguajes de tercera y cuarta generación (3GL y 4GL) asistidos con entornos gráficos y funciones CASE especializadas. Actualmente se está trabajando para avanzar a la 'programación con modelos' o desarrollo mediante modelos, que ya ha mostrado sus primeros frutos. En este capítulo se muestra como las actividades de construcción de software progresivamente se han venido apartando de la máquina y de la tecnología para centrarse en las actividades de ingeniería y arquitectura de software.

La evolución de la construcción de software ha sido abordada desde diferentes puntos de vista. Una visión interesante, en este sentido, es la de Frankel [1] quien presenta las evolución de esta práctica en tres etapas grandes etapas. La primera (centrada en la máquina) representa los inicios de la programación binaria y la aparición del lenguaje assembler hacia el final de los años 50 y comienzos de los 60 [9]; la segunda (centrada en la aplicación) caracterizada por la evolución del lenguaje assembler y la aparición de los lenguajes de segunda y tercera generación (COBOL) por los años 60 y 80 [9]; y la tercera, a partir de los años 90, centrada en la empresa, destaca los lenguajes de ambientes gráficos y cuarta generación.

1.1 COMPUTACIÓN CENTRADA EN LA MAQUINA

La computación centrada en la máquina se basaba en los lenguajes de programación en binario y ensamblador (assembler). La programación en binario consistía en escribir instrucciones código en 1s y Os, mediante patrones de bit exclusivos de cada tipo de CPU de cada máquina. De manera que la codificación era excesivamente dispendiosa y se limitaba a un reducido conjunto de tareas, aunque permitía optimizar la memoria secundaria y la memoria de procesamiento [10]

La aparición del lenguaje assembler mitigó algunas de las desventajas de la programación binaria y se extendió el tiempo de vida útil de la computación centrada en la máquina. Assembler disminuyó el tiempo de desarrollo, los costos de producción y la probabilidad de cometer errores debidos a la programación binaria. En consecuencia fue posible escribir tareas más complejas (aplicaciones) y aumentó la calidad de los programas.

1.2 COMPUTACIÓN CENTRADA EN LA APLICACIÓN

La necesidad de escribir aplicaciones que resolvieran problemas de negocios complejos, se vio apoyada con los emergentes lenguajes de programación de tercera generación (3GL) tales como FORTRAN y COBOL [11] Estos lenguajes elevaron el nivel de las de las primarias instrucciones manuales de procesador. La invención de los compiladores hizo más eficiente el desempeño de los 3GLs que traducían las instrucciones de alto nivel en instrucciones de procesador nativo o código de máquina.

La introducción de los lenguajes estructurados (C y Pascal), los depuradores de alto nivel y la reducción del número de líneas, mejoró la longevidad de los programas escritos en 3GLs, pues la arquitectura de computador dejó de ser un factor de obsolescencia de los programas [12]

Cuando los lenguajes estructurados empezaron a experimentar problemas, evolucionaron a lenguajes orientados a objetos básicos. Estos a su vez se perfeccionaron con máquinas virtuales que hizo posible la portabilidad de aplicaciones a diferentes ambientes de computación.

1.3 COMPUTACIÓN CENTRADA EN LA EMPRESA

El incremento en el nivel de automatización de la empresa promovió la integración de las "islas de automatización" para evitar la superposición de la funcionalidad, que creaba duplicación de la información, y mejorar el uso de los recursos computacionales. Adicionalmente, se desarrollaron un conjunto de conceptos y tecnologías que favorecieron la computación centrada en la empresa, tales como: Desarrollo Basado en Componentes, Diseño de Patrones, Computación Distribuida, Middleware aplicada a la plataforma, Middleware aplicada a la programación, especificación declarativa, arquitectura de la empresa y separación de intereses, Integración de aplicaciones empresariales y Diseño por Contrato.

El *Desarrollo Basado en Componentes* se basa en las mejores experiencias de los procesos de producción industrial; promueve la práctica de ensamblar aplicaciones a partir de componentes intercambiables [13], [14]

El *Diseño de Patrones*, también es un concepto del proceso de producción industrial que contribuye al mejoramiento de la productividad y calidad del desarrollo. Bajo esta perspectiva los programadores pueden reutilizar patrones que otros han creado y validado [15]

La *Computación Distribuida* permitió compartir primero recursos de hardware costosos, y después recursos de software. En esta línea apareció la computación cliente servidor, que posteriormente evolucionó al paradigma peer-to-peer [16]

El *middleware* elevó el nivel de abstracción de la plataforma. La comunicación evolucionó gradualmente hasta ser asumida por el sistema operativo distribuido. Después elevó el nivel de abstracción hasta convertirse en middlewares con servicios potentes, tales como CORBA, J2EE, .NET y MOM – Middleware Orientado a Mensajes [17], [18]

El *middleware* elevó el nivel de abstracción de la programación. Algunos middlewares proveen servicios independientes del sistema operativo y/o lenguaje de programación; CORBA, por ejemplo, posee un servicio de concurrencia que provee una API que las aplicaciones pueden invocar para activar y liberar un bloqueo [17], [18]

La *especificación declarativa*, mejora la productividad y la calidad porque es otra forma de reutilizar lógica pre-programada y pre-validada [19], [20]

La *arquitectura de la empresa y la separación de intereses*, pretende organizar sistemas complejos separando los intereses (concerns). La separación de intereses tiende a localizar los cambios de un aspecto del sistema, de manera que cuando un cambio de un aspecto impacta sobre otros aspectos, la separación de intereses facilita rastrear el impacto [21]

La *integración de aplicaciones empresariales EIA*, hizo posible el diseño de sistemas empresariales que estuvieran naturalmente bien-integrados. Utilizando la arquitectura empresarial es posible crear nuevos sistemas software que no tengan islas funcionales, pero es necesario integrar dentro de la arquitectura de la empresa las islas existentes, los sistemas legados y aplicaciones empaquetadas [22]

El *diseño por contrato DBC*, es un enfoque para construir software fiable que consiste en hacer un contrato explicito y formal. Los fundamentos básicos de diseño por contrato fueron formulados por Dijkstra en los años 70 [23], [24]

En conclusión, el desarrollo de la computación se ha concentrado en elevar el nivel de abstracción de los lenguajes de programación, de modo que los desarrolladores se centren más en las tareas de ingeniería y de arquitectura del sistema y menos en los aspectos tecnológicos. El desarrollo centrado en los modelos del sistema se perfila como el siguiente nivel de abstracción. MDA está aprovechando la tecnología de la computación centrada en la empresa para definir un conjunto de estándares que orientan el desarrollo de software y MDSD se ocupa de construir herramientas que implementan la idea de desarrollo conducido por modelos.

2. DIFERENCIA ENTRE MDA Y MDSD

Aunque MDA y MDSD son comunidades diferentes comparten el mismo propósito: la utilización de modelos para desarrollar software. Mientras que MDSD se ocupa la utilización de los modelos para el desarrollo de software, MDA se enfoca en definir los estándares para desarrollar software haciendo uso intensivo modelos. Se pudiera decir que MDA es la estandarización de MDSD.

La diferencia ente MDA y MDSD ha sido expuesta por diversos autores e investigadores. Mellor [25] plantea que MDA se puede percibir como un conjunto de estándares cuya aplicación concreta es el desarrollo de software dirigido por modelos, y la principal actividad de MDSD es la construcción de herramientas que privilegian el desarrollo mediante la utilización de modelos. Selic [26] afirma que los métodos y herramientas que soporten tanto a MDA como a MDSD deben especificar transformaciones automáticas entre diversos modelos, diferenciando así los dos términos. Frank [27] plantea que el desarrollo de software guiado por modelos, tiene que ver con los modelos usados por arquitectos y programadores para alimentar generadores de código, son técnicas que también están bajo el marco de MDSD, pero que no pueden ser correctamente denominadas MDA. Dirk [28] afirma que el desarrollo de software conducido por modelos MDSD, reúne el conjunto de tecnologías que usan los modelos para desarrollar, en consecuencia MDA, es la arquitectura definida por OMG para soportar MDD.

En conclusión, aunque MDA y MDSD son enfoques diferentes, los dos privilegian el uso del modelo como 'lenguaje de desarrollo'. MDSD ha existido mucho antes que MDA como práctica de desarrollo y como investigación para generar métodos, técnicas, herramientas y constructos teóricos basados en modelos. MDA por su parte ha hecho un gran esfuerzo por estandarizar la amplia pero poco uniforme actividad de desarrollar software conducido por Modelos MDSD.

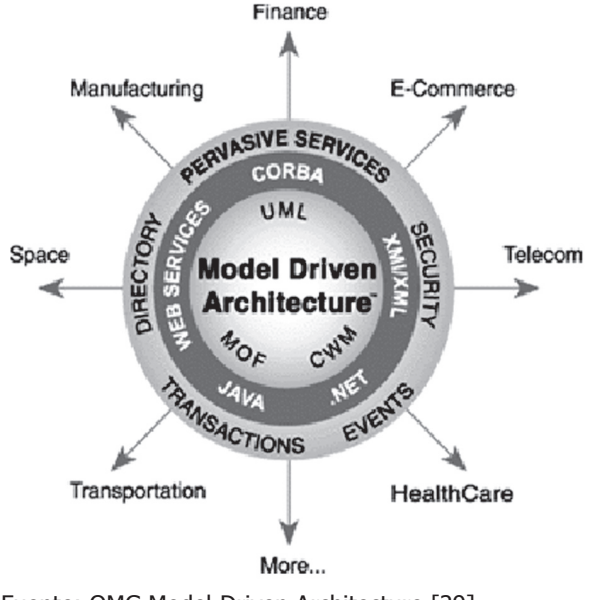
3. FUNDAMENTACIÓN TEÓRICA GENERAL

De lo expuesto anteriormente se desprende que MDA y MDSD son enfoques complementarios que usan el modelo como directriz del desarrollo. MDA por ser un conjunto de estándares promovidos por OMG, se caracteriza por tener bases teóricas fuertes, mientras que MDSD por ser una iniciativa en manos de una numerosa comunidad de investigación, se centra prioritariamente en el desarrollo de herramientas que facilitan la automatización del desarrollo.

3.1 MDA EN EL CONTEXTO DE ESTÁNDARES DE OMG

En este entorno, los modelos independientes de la plataforma que incluyen los estándares de modelado de OMG, se pueden desarrollar utilizando plataformas abiertas o propietarias como CORBA, Java, .NET, XMI/XML y plataformas basadas en la Web (ver figura 1). Sin embargo, la tendencia dominante para el modelo dependiente de la plataforma es la utilización de java, debido a los recursos y facilidades de esta plataforma. Adicionalmente, a medida que surgen nuevas tecnologías, MDA desarrolla rápidamente las especificaciones pertinentes, de forma que facilita el proceso de integración. Se concluye que MDA es más que un middleware, ofrece una solución integral y estructurada para la interoperabilidad y portabilidad de aplicaciones en el futuro [29], [30]

FIGURA 1. MDA en el ámbito de los estándares de OMG



Fuente: OMG Model-Driven Architecture [29]

De los estándares que conforman el núcleo de MDA, el Common Warehouse Model CWM requiere atención especial ya que proporciona las pautas para la gestión e integración datos, como se muestra en la siguiente sección.

3.2 MODELO COMÚN DE BODEGAJE CWM

Es un estándar de la OMG para bodegas de datos, que define la especificación de modelado de metadatos para objetos relacionales, no-relacionales, multidimensionales y otros objetos del ambiente de bodega de datos. Abarca el ciclo de vida completo de diseño, construcción y gestión de aplicaciones y el apoyo a la gestión del ciclo de vida [1]

CWM especifica las interfaces que habilitan el intercambio de metadatos de bodegas de datos e inteligencia de negocios entre artefactos de bodegaje de datos, en ambientes distribuidos heterogéneos, tales como herramientas, plataformas y repositorios [31], [32], [33]

CWM se basa en tres estándares que conforma el núcleo de la arquitectura del repositorio de metadatos de OMG [34]:

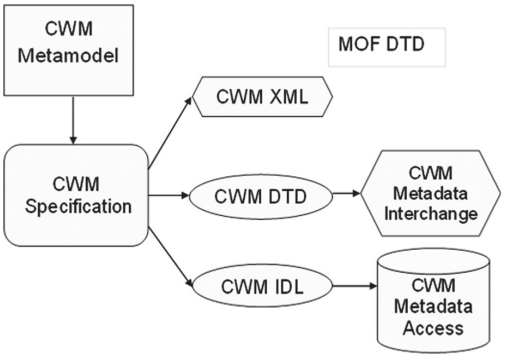
- Lenguaje de modelado unificado (UML). Lenguaje prioritariamente gráfico que permite definir y representar modelos y meta-modelos del sistema, modelos de diseño y modelos de información.

- Facilidad meta-objeto (MOF). Es un estándar que permite definir modelos de metadatos y modelos de modelos. Provee herramientas con interfaces programables para almacenar y acceder metadatos en un repositorio.

- Intercambio de metadatos XML (XMI). Estándar que permite intercambiar metadatos presentados como archivos en formato XML.

Una especificación CWM incluye componentes básicos y un conjunto de metamodelos. Los componentes básicos de de CWM se muestran en la figura 2 y son los siguientes: metamodelo CWM, formato de intercambio para metadatos de bodegaje compartidos (CWM DTD), formato de intercambio para el metamodelo (CWM XML) y API de acceso para metadatos de bodegaje compartidos (CWM IDL) [35]

FIGURA 2. Especificación Common Warehouse Metamodel CWM



Fuente: adaptado de POOLE, John y otros. CWM Enablement Showcase [35]

Los metamodelos que incluye una especificación CWM representan recursos de datos, análisis, aspectos de gestión de bodegaje y los componentes fundamentales de un entorno de bodegaje de datos e inteligencias de negocios. Además, CWM define metamodelos principalmente para los siguientes dominios [30], [36]:

- Modelo de objetos (un subconjunto de UML)
- Recursos de datos relacionales
- Recursos de datos registro
- Recursos de datos multidimensionales
- Recursos de datos XML
- Transformaciones de datos
- OLAP (On-line Analytical Processing)
- Minería de Datos (Data Minning)
- Visualización de información
- Nomenclatura de negocios
- Procesos de bodegas de datos
- Operaciones de bodegas de datos

3.2.1 Metamodelo CWM

El metamodelo CWM está conformado por varios sub-metamodelos que representan metadatos comunes de bodegaje (figura 3), en las principales áreas de interés de bodegas de datos e inteligencia de negocios [36], [37]:

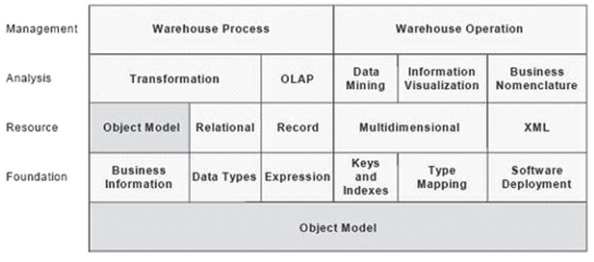
- *Recursos de Datos*. Incluye metamodelos que representan recursos de orientación a objetos, relacionales, de registro, multidimensionales y XML.

- *Análisis de Datos*. Considera los metamodelos que representan transformaciones de datos, procesamiento analítico en línea (OLAP), minería de

datos, información de visualización y nomenclatura de negocios.

- *Administración de Bodegas de Datos*. Abarca los metamodelos que representan procesos de bodegas de datos y resultados de operaciones de bodegas de datos.

FIGURA 3. Metamodelo CWM



Fuente: OMG. CWM Specification [36]

Para controlar la complejidad, mejorar la comprensión y soportar la reutilización, CWM utiliza los siguientes paquetes: Object Model Package, Foundation Package, Resource Package, Analysis Package y Management Package. Cada paquete contiene clases, asociaciones y otros elementos [36], [37]

Para facilitar su comprensión, la especificación de paquetes de metamodelos CWM se representa mediante UML. Estas representaciones incluyen nombres, clases, atributos, tipos de datos, asociaciones, extremos de asociación, referencias, restricciones, diagramas de instancias y modularidad [37], [38]

CWM Metadata Interchange Patterns Specification es un metamodelo que extiende el estándar CWM sin alterar ninguna de las implementaciones existentes y permite a los usuarios utilizar los modelos de bodega de datos definidos por CWM y agrega semántica para obtener prestaciones adicionales [39]

3.2.2 CWM para Modelado de Transformaciones

CWM apoya las transformaciones de datos dirigidas por modelos. Para esto dispone de metamodelos para bases de datos relacionales, bases de datos multidimensionales y estructuras de registros y XML. También contiene un conjunto de metamodelos que describen reglas de procesamiento de diferentes formas: Procesamiento analítico en línea (OLAP), Minería de Datos y Transformación de datos [1]

El metamodelo de transformación CWM define constructos para especificar reglas de transformación. Con base en éste, el usuario describe las transformaciones en términos de los modelos de datos fuente y destino, que son compatibles con todos los metamodelos de datos CWM. OMG aplica mapeos XMI MOF-XML al metamodelo de transformación CWM para producir una DTD que constituye una sintaxis concreta para la representación de reglas de transformación. Igualmente, aplica mapeos MOF-IDL para producir CORBA IDL para la representación de reglas de transformación como objetos CORBA. Finalmente, aplica mapeo JMI MOF-Java al metamodelo para generar APIs Java que representan las reglas de transformación como objetos Java.

Las principales estrategias de implementación de modelos de transformaciones CWM son [1]:

- Utilizar una herramienta repositorio MOF que genere código para el manejo de reglas de transformación de la misma manera que lo hace para manejar otros tipos de metadatos.

- Construir un generador que lea un conjunto de reglas de transformación (M1) y genere código de transformación y ejecute las reglas en datos (M0).

- Crear un transformador genérico (máquina virtual) que lea las reglas de transformación (M1) dinámicamente en tiempo de ejecución y ejecute transformaciones en los datos (M0).

El metamodelo de transformaciones CWM soporta transformaciones entre cualquier par de modelos de datos compatibles con CWM. Para esto define los elementos principales de los metamodelos de datos, tales como el metamodelo XML y las subclases de un conjunto general de superclases CWM.

3.3 ARQUITECTURA CONDUCIDA POR MODELOS MDA

La evolución del desarrollo de software, se ha visto influenciada por el continuo ascenso acenso del nivel de abstracción de los lenguajes de programación. Del lenguaje binario se pasó al código ensamblado, de éste, a los lenguajes de tercera generación en modo texto, posteriormente estos fueron mejorados con ambientes gráficos y con ayudas inteligentes de edición y compilación, y paralelamente aparecieron los potentes 4GL para gestionar bases de datos. La actual actividad tecnológica y de investigación está dirigida a dar el siguiente gran paso; el desarrollo centrado modelo.

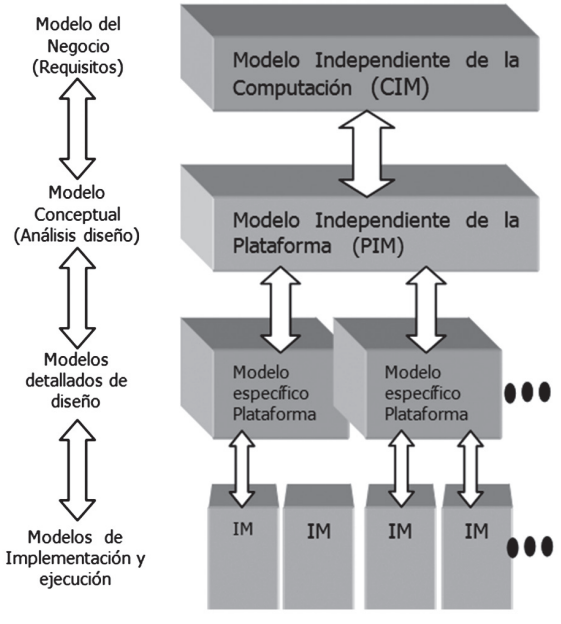
Los entornos de desarrollo de software centrados en modelado proveen un 'lenguaje' muy próximo al lenguaje del desarrollador y del cliente. Esta estrategia aleja al desarrollador de las prácticas tradicionales de codificación en lenguajes de programación orientados exclusivamente a la solución.

Los soportes teóricos y prácticos para este cambio de nivel en los lenguajes de desarrollo de software los proporciona MDA y MDSD. En este sentido, Brown [39] sostiene que modelo, modelaje y MDA son los fundamentos de la tendencia conocida como MDD, donde los modelos son utilizados para razonar acerca del dominio del problema y del dominio de la solución.

Los principales aspectos de MDA, según Debnath [41], son los siguientes:

- Cambia el foco del desarrollo de software, de la escritura de código a la construcción de modelos.
- La idea central de MDA (ver figura 4) es separar la especificación de la funcionalidad del sistema de los detalles de su implementación en una plataforma específica [42]
- Su principal meta es lograr portabilidad, interoperabilidad y reutilización mediante la separación de los aspectos de arquitectura.
- Considera que todos los artefactos son modelos, es decir: especificación de requisitos, descripciones de arquitectura, descripciones de diseño y código.
- Es un enfoque conducido por modelos porque provee los medios para usar modelos que guíen el curso del entendimiento, diseño, construcción, despliegue, operación, mantenimiento y modificación.

FIGURA 4. Modelos por niveles y transformaciones MDA



Fuente: Adaptado de Brown [40]

- La 'transformación automática' es una de las nociones centrales de MDA. Ella describe como un modelo en un lenguaje fuente (modelo fuente) se puede transformar en uno o más modelos en un lenguaje objeto (modelo objeto).

El OMG es la organización que establece las directrices para el desarrollo de software centrado en modelos y están plasmados en MDA. Los cuatro principios fundamentales de MDA, según son Brown [40], son los siguientes:

- Modelos bien definidos (sintáctica y semánticamente). Esto constituye el principal aspecto para comprender el sistema y para implementar soluciones empresariales escalables.
- Un sistema se puede construir mediante la elaboración de modelos, el establecimiento de las transformaciones entre modelos y la organización del trabajo en una arquitectura de niveles y transformaciones.
- Los metamodelos constituyen la estructura formal que facilita la transformación e integración entre modelos, y son fundamentales para la automatización mediante herramientas.
- Para aceptar y adoptar el desarrollo centrado en modelos se requiere la formulación de estándares industriales, para brindar facilidades a los clientes y habilitar una clara competencia entre vendedores.

Uno de los aspectos centrales del marco de trabajo de MDA es la noción de transformación automática. Ésta especifica como transformar un modelo escrito en un lenguaje fuente (modelo fuente) en un lenguaje objeto (modelo objeto). Las vistas de especificación de un sistema (figura 4) facilitan la transformación de modelos más abstractos a modelos menos abstractos. Ellas se describen a continuación, según el punto de vista de Debnath [41]

Modelo Independiente de la computación CIM. Este es el modelo de mayor abstracción, pues oculta cualquier detalle relacionado con la solución (sistema software). Describe el sistema desde el punto de vista del dominio del negocio, usando un lenguaje cercano a los especialistas en dominio del negocio, por lo que también se le denomina Modelo del Dominio.

Modelo independiente de la plataforma PIM. Este modelo presenta una vista del sistema, pero independiente de cualquier plataforma en la que se vaya a implementar la solución. Posee características tales que de él se puede derivar especificaciones para implementarlas en diferentes plataformas.

Modelo específico de la plataforma PSM. Es un punto de vista orientado a una plataforma software específico. Esta vista incluye especificaciones del PIM combinadas con detalles de una plataforma particular que describe como el sistema usará tal plataforma para implementar la solución.

Todo marco de desarrollo de software incluye implícita o explícitamente un proceso de desarrollo de software. MDA sugiere, con base en la estructura de niveles y transformaciones, el siguiente proceso [41]

- Captura del conocimiento del negocio y los requisitos del usuario para ser representados en un CIM.
- Construcción de un modelo del sistema (PIM) que satisface los requisitos expresados en el CIM. El PIM representa el sistema sin tener en cuenta la plataforma tecnológica específica en la que se implementará el sistema.
- Transformación del PIM en uno o más PSM's, con la ayuda de una herramienta MDSD automática que utiliza un conjunto de reglas de transformación.
- Obtención del código a partir de un PSM. Dado que un PSM es aún muy abstracto para compilarlo en un lenguaje de programación, se requiere otro conjunto de reglas de transformación y una herramienta MDSD para generar el código a partir del PSM.

La automatización del proceso anterior implica: la utilización de un lenguaje formal de escritura para los modelos, establecer detalladamente las correspondencias los modelos, formalizar las reglas de transformación y construir herramientas que implementen automáticamente tales transformaciones.

Lastimosamente, este proceso no considera la posibilidad de transformar el CIM en un PIM. Aunque el CIM es bastante abstracto, parece necesario establecer un procedimiento que permita construir el PIM a partir del CIM, con la ayuda de una herramienta MDSD y un conjunto de reglas de transformación.

En definitiva, como plantea Debnath [41], en MDA, los modelos son el núcleo del esfuerzo de desarrollo. Ellos se debieran ver como código del más alto nivel (especificaciones de modelo), que juegan un papel central en el desarrollo, evolución y mantenimiento del software.

3.4 DESARROLLO DE SOFTWARE CONDUCIDO POR MODELOS MDSD

Como ya se ha discutido, el Desarrollo de Software Conducido por Modelos MDSD puede llegar a ser el siguiente paso de la evolución de las actuales prácticas de programación. Esta idea se respalda en la consolidación de de UML como lenguaje de modelado, la masificación de las prácticas modelado y la aparición de MDA.

Sin embargo, es necesario diferenciar entre desarrollo de software basado en modelos y desarrollo de software conducido por modelos. El primero implica el uso de modelos como medio de documentación del desarrollo, sin que éstos lleguen a constituirse en código como tal. Mientras que el segundo, considera que el modelo es código de alto nivel, dado que su implementación es automática, es decir su compilación final genera código en lenguajes 3GL [43]

La principal diferencia entre estos dos enfoques de desarrollo está centrada en la semántica de las palabras *basado* y *conducido*; las implicaciones de uno y otro término, definen los siguientes aspectos [43]:

- Los *sistemas son dinámicos*, de manera que ellos pueden llegar a requerir cambios significativos, especialmente en las primeras etapas del ciclo de vida.
- La 20, en estos términos, es una tarea compleja que requiere ser adaptada continuamente y que dependiendo del nivel de detalle pudiera llegar a ser inconsistente.
- Cuando los *modelos se utilizan como documentación* solo contribuyen parciamente al desarrollo, pues únicamente son un medio de interpretación que conducen a obtener manualmente el código del programa. Por este motivo muchos programadores, consideran que la actividad de modelado tan solo es una carga adicional de trabajo.
- En el enfoque conducido por modelos, el *modelo* no constituye documentación sino que se le considera código de alto nivel, que solo requiere una transformación automática para obtener código en lenguajes tradicionales de implementación.
- El *dominio del problema* es esencial para modelado, especialmente en los procesos de producción automática. Por esta razón, un objetivo de MDSD es encontrar abstracciones de dominio específicas que faciliten el modelado formal.
- El hallazgo de *abstracciones específicas del dominio* incrementa la productividad, facilita la automatización de la producción de software, incrementa la calidad y mantenibilidad y facilita la comunicación entre el ingeniero de software y los expertos del dominio.

- Para aplicar exitosamente el concepto de *modelo especifico del dominio*, se requiere:
  - Lenguajes específicos del dominio que permitan la formulación de modelos reales.
  - Lenguajes que puedan expresar las transformaciones requeridas de modelos a código.
  - Compiladores, generadores o transformadores que puedan ejecutar las transformaciones para generar código ejecutable en diversas plataformas.

- Finalmente, aunque en el contexto de MDSD se usan modelos gráficos, los modelos textuales son igualmente válidos. Unos y otros se trasladan a código fuente en lenguajes de programación para su posterior compilación y ejecución.

En definitiva, MDSD sostiene que no basta con escribir modelos como mera documentación, sino que el propósito final es que los modelos lleguen a constituir el código de mayor nivel, de forma que pueda ser escrito, modificado y compilado a lenguajes 3GL. Dirk [28] respalda esta idea; afirma que modelar es una forma de describir el software que con frecuencia está más cerca del dominio del experto que del código. Así el experto es capaz de describir conceptos y sus relaciones de una forma visual intuitiva, por ejemplo usando UML.

4. HERRAMIENTAS PRÁCTICAS Y CONCEPTUALES

La investigación en MDSD ha producido diversas herramientas; algunas para implementar el desarrollo guiado por modelos y otras para generar elementos teóricos y conceptuales para apoyarlo. En este capítulo se presentan algunas de las propuestas mas interesantes en esta área.

4.1 HERRAMIENTAS CONCEPTUALES

El concepto de transformación es un elemento central de MDSD y las hay de diversas tipos. Un enfoque interesante de diferentes tipos de transformaciones y sus características se presentan en Metzger [44] Las transformaciones más conocidas son las verticales que convierten un modelo de alto nivel de abstracción en uno de más bajo nivel. En este trabajo se presenta un esquema de clasificación detallado conveniente para que el lector obtenga un buen conocimiento de cómo tales tipos de transformaciones pueden abordar los problemas mas frecuentes de MDSD.

La tecnología de agentes se ha estado usando ampliamente para construir aplicaciones industriales y comerciales complejas. En este sentido Zhu [45] sostiene que esta práctica con frecuencia carece de rigor y de un lenguaje que le soporte. Por esta razón propone el lenguaje CAMLE y una herramienta automática de modelado. El lenguaje se concentra en la consistencia de restricciones sobre modelos gráficos.

Las transformaciones gráficas de modelo son una alternativa adecuada para representar las transformaciones a alto nivel. Grunske [46] afirma que este tipo de transformaciones son necesarias para implementar MDA. Por esta razón propone un formalismo gráfico basado en sistemas de transformación gráfica y en reglas de transformación gráfica. Esta propuesta se perfila como una de las posibilidades más relevantes para formalizar las transformaciones de modelo con el propósito de automatizar las prácticas de MDSD.

El nivel específico de MDA se refiere al modelo que describe aspectos dependientes del ambiente de implementación, por tanto, los conceptos de plataforma y modelo de plataforma juegan un papel central en el proceso de transformación de modelos. Atkinson [47] afirma que estos conceptos no están claramente definidos y en consecuencia hace una amplia disertación de lo que se debe entender como plataforma y como modelo de plataforma. Adicionalmente describe el lenguaje de dominio específico DSL, sus pros y sus contras.

Dirk [28] propone los conceptos de Desarrollo Generativo Conducido por Modelos, para soportar la evolución de granularidad fina. Lo anterior para resolver el problema identificado en MDA de tener que regenerar y reinicializar el sistema cuando se requieren adaptar los modelos de un sistema existente. Este problema es atribuido a la fuerte separación entre el ambiente de modelado y el ambiente de ejecución. El trabajo se centra en adicionar o reemplazar clases del modelo.

4.2 HERRAMIENTAS PRÁCTICAS

La calidad, es un factor relevante en los sistemas de seguridad crítica; requiere un manejo muy cuidadoso. Para colaborar en el logro de este aspecto, Jürjens [48] propone el concepto de máquinas UML, como notación formal que refleja las propiedades de la semántica de ejecución para requisitos críticos. Esta es la base de una herramienta que se construyó con el propósito de gestionar el desarrollo del sistema, pero especialmente de los requisitos críticos.

Los concerns (o intereses) son un concepto que surgió para tratar de gestionar la superposición de funcionalidades debidas a cruces entre diversos intereses de los clientes en un sistema; es decir los cruces

transversales (cross-cutting concerns). Hammouda [49], propone el concepto de patrón aspectual para representar los aspectos en diferentes niveles de abstracción. Adicionalmente presenta una herramienta que permite procesar los intereses y aspectos para generar el modelo y para hacer verificación y rastreo de aspectos.

El refactoring es un enfoque importante para mejorar la estructura interna de un sistema, manteniendo su comportamiento externo. Zhang [50] acuñó el concepto de 'refactoring de alto nivel' y lo aplicó en el diseño de un motor de transformación de modelos. Este motor permite hacer transformaciones en un dominio específico, de manera que facilita el trabajo de refactoring.

Otras herramientas interesantes en el entorno de MDA y MDSD son las siguientes: Automatically Discovering Transitive Relationships in Class Diagrams [51], un motor de transformación de modelos: A Testing Framework for Model Transformations [52] y generación de código para middleware: Parallax – An Aspect-Enabled Framework for Plug-in-Based MDA Refinements Towards Middleware [53]

5. TENDENCIAS ACTUALES

A pesar de la gran cantidad recursos, tiempo y esfuerzos dedicados a la investigación de MDA y MDSD, la generación de lenguajes basados en modelo para el desarrollo de software apenas está en su fase inicial. Para avanzar en este proyecto, la investigación y la tecnología tendrían que abordar trabajos como los siguientes:

- Continuar con el desarrollo de conceptos y teorías que ayuden a formalizar los modelos en sus diferentes niveles de abstracción.
- Desarrollar teórica y tecnológicamente el nivel CIM de MDA. Este nivel ha sido poco considerado por la comunidad de investigación y aún por el OMG.
- Establecer en forma explícita las transformaciones entre modelos y las reglas necesarias para transformar modelos fuentes en modelos objetos.
- Articular herramientas MDSD que realizan transformaciones parciales, en herramientas integradas que automaticen el desarrollo de software

6. CONCLUSIONES Y TRABAJOS FUTUROS

El presente trabajo se ha realizado, en el marco del proyecto de investigación titulado "Construcción de un Proceso de Desarrollo de Software con base MDA y MDSD", con el propósito de establecer el estado del conocimiento de MDA y MDSD. Así que las principales conclusiones, que han soportado el desarrollo de tal proyecto de investigación, son las siguientes:

- La Evolución del foco de la computación muestra que la práctica de desarrollo de software ha estado orientada a liberar al desarrollador de detalles tecnológicos (hardware y código) para concentrar su atención en los asuntos de ingeniería.
- El propósito de MDA y MDSD es el desarrollo mediante modelos, en contraste con las prácticas anteriores, en las que los modelos sólo tenían propósitos de documentación.
- Se estableció plenamente la diferencia entre MDA y MDSD. MDA se centra en los aspectos teóricos y de estandarización de la propuesta y MDSD privilegia la construcción de herramientas conceptuales y prácticas orientadas a producir software en forma automática.
- MDA conforma un núcleo de estándares de OMG (UML, MOF y CWM entre otros), los cuales la complementan y facilitan su real implementación.
- Dos de los conceptos centrales de desarrollo conducido por modelos, son el modelo y la transformación automática de modelos. El primero guía todo el proceso de desarrollo y el segundo asegura su automatización.

Estos aspectos concluidos, contribuyen efectivamente a la orientación y desarrollo del proyecto de investigación antes mencionado. Además, el éxito de este proyecto depende de algunas de las siguientes actividades que se abordarán próximamente:

- El trabajo necesario para conseguir el objetivo de elevar el nivel del lenguaje de desarrollo de lenguajes de programación a lenguajes basados en modelos, aunque ha sido arduo y se han logrado avances significativos, todavía no muestra una herramienta generalmente aceptada.
- El propósito de elevar el nivel de desarrollo no tiene vuelta atrás, ignorar esta realidad implica someterse al atraso teórico y tecnológico.
- Se debe seguir trabajando en el propósito de definir un proceso que integre los conceptos de MDA y MDSD.
- Es necesario hacer un estudio profundo de las herramientas que automatizan a MDA y MDSD.

- Se debe trabajar mucho en el empeño de definir los modelos que constituyen cada nivel de MDA, las transformaciones y las reglas de transformación

7. REFERENCIAS BIBLIOGRÁFICAS

[1] FRANKEL, David. Model Driven Architecture – Applying MDA to Enterprise Computing. USA: Wiley Publishing, Inc. 2003.

[2] LAUDON, Kennet and LAUDON, Jane. Sistemas de información Gerencial: Administración de la Empresa Digital. Pearson Ed. 2008.

[3] ER 2010 Workshops ACM-L, CMLSA, CMS, FP-UML, SeCoGis, WISM. Advances in Conceptual Modeling Aplications and Challenges. Vancouver, Canada, 2010.

[4] OMG Object Management Group. Unified Modeling Language, Infrastructure. Version 2.4.1. Standard document, 2011 (en línea). http://www.omg.org/spec/UML/2.4.1/Infrastructure.

[5] FAVRE, Liliana. Model Driven Architecture for Technologies. DOI: 10.4018/978-1-61520-649-0.ch001. Argentina, 2010.

[6] FRANCE, Robert y RUMPE, Bernhard. Model Driven Devepment of Complex Software: A Research Road Map. Proceeding FOSE'07 2007 Future of Software Engineering. Doi 10.1109/FOSE.207-14. Washington: IEEE Computer Society, 2007.

[7] [02] HECKEL, Oscar y LOHMANN, Marc. Towards Model Driven Testing. Elseiver – Electronics Notes in Theoretical Computer Science. Germany: Elsevier, 2011 (en línea). (http://www.sciencedirect.com/science/article/pii/S1571066104810235)

[8] PASTOR, Oscar y otros. Model-Driven-Development. En Springer Verlag Computer Science - Informatik-Spektrum, Vol. 31, No. 5, p. 394-407. Berlín, 2008.

[9] FREED, Leonard. The History of Computers. Ziff-Davis Press, Emeryville (Ca), 1995.

[10] BECK, Leland. System Software: An Introduction to Systems Programming. Addison Wesley, 1996.

[11] YANG, Hongji y WARD, Martin. Successful evolution of Software System. Architect House Computing library. Library of Congress Cataloging-in-Publication Data. 2003.

[12] SHAMMAS, Namir. Introducing C to Pascal Programmers. Ed. Wiley, 1988.

[13] HERSUM, P. y SIMS O. Business Component Factory: A Comprehensive Overview of Component Based Development for the Enterprise. New York: John Wiley & Sons, 2000.

[14] EELES, Peter y CRIPPS, Peter. The Process of Software Architecting. USA: Addison-Wesley. 2009.

[15] BUSCHMANN, Frank; HENNEY, Kevlin y SCHMIDT, Douglas C. Pattern-Oriented Software Architecture, Volume 4: A Pattern Language for Distributed Computing. New York: John Wiley & Sons. 2007.

[16] TANEMBAUM, Andre W. y VAN STEEN, Maarten. Distributed Systems: Principles and Paradigms. USA: Pearson Prentice Hall. 2007.

[17] Deepak. ALUR, John. CRUPI, and Dan. MALKS, Core J2EE Patterns: Best Practices and Design Strategies, Sun Microsystems Press, 2001.

[18] Wolfgang. EMMERICH, Mikio. AOYAMA, J. SVENTEK. The impact of research on middleware technology. ACM SIGSOFT Software Engineering Notes, Vol. 32, No. 1, 2007.

[19] GRAY, Jim y REUTER, Andreas. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993.

[20] MANJARRÉS, Roberto A. Generación de la Especificación Declarativa a Partir de la Especificación Gráfica de un Metamodelo. Tesis de Grado Maestría en Ingeniería de Sistemas. Medellín: Universidad Nacional de Colombia, 2010.

[21] ROSS, Jeanne; WEILL, Peter; ROBERTSON, David. Enterprise architecture as strategy: creating a foundation for business execution. Harvard Business Scholl Press, 2010.

[22] MENTZAS, Gregoris y FRIESEN, Andreas. Semantic Enterprise Application Integration for business processes. Service-Oriented Frameworks. USA: Business Science Reference IGI Global. 2010.

[23] DIJKSTRA, Edsger. A Discipline of Programming. USA: Prentice Hall, Inc. 1976.

[24] MEYER, Bertrand. Touch of Class: Learning to Program Well with Objects and Contracts. Berlín: Springer, 2009.

[25] MELLOR, Stephen; CLARK, Anthony and FUTAGAMI, Takao. Model-driven development – Guest editor's introduction. En Revista IEEE Software, Vol. 20 No. 5, p. 14-18, Septiembre-Octubre 2003.

[26] SELIC, Bran. The Pragmatics of Model-Driven Development. En Revista IEEE Software, Vol. 20 No. 5, p. 46-51, Septiembre-Octubre 2003.

[27] FRANK, Karl y otros. Reaching Toward MDA. OMG White Paper, 2008 (en línea). http://www.omg.org

[28] DIRK, Theo; PETTERSEN, Jan; PRINZ, Andreas y WORTMANN, Hans. Supporting fine-grained generative model driven evolution. Regular Paper, 2010 (en línea). http://www.springerlink.com.

[29] OMG Object Management Group. Model-Driven Architecture Home Page, 2011 (en línea). http://www.omg.org/mda/index.htm

[30] POOLE, John. Model-Driven Architecture: Vision, Standards and Emerging Technologies, 2001 (en línea). http://www.cwmforum.org/Model-Driven+Architecture.pdf

[31] MILLER, Frederic; VANDOME, Agnes. Common Warehouse Metamodel CWM. VDM Verlag, Alphascript Publishing, 2010.

[32] CWMFORUM. What is the Common Warehouse Metamodel CWM, 2011 (en línea). http://www.cwmforum.org/about.htm

[33] KIMBALL, Ralph y ROSS, Margy. Model-Driven Data Warehousing. New York: John Wiley & Sons Inc. 2011.

[34] BARRY, Douglas K. Common Warehouse Metamodel CWM, 2011 (en línea). http://www.service-architecture.com/web-services/articles/common_warehouse_meta-model_cwm.html

[35] POOLE, John; CHANG, Dan; TOLBERT, Douglas y MELLOR, David. Common Warehouse Metamodel Developer's Guide. New York: John Wiley & Sons Inc. 2008.

[36] OMG. Common Warehouse Metamodel CWM Specification. Versión 1.1. Volumen 1, 2003 (en línea). http://www.omg.org/spec/CWM/1.1/PDF

[37] POOLE, John; CHANG, Dan; TOLBERT, Douglas y MELLOR, David. Common Warehouse Metamodel. An Introduction to the Standard for Data Warehouse Integration. New York: John Wiley & Sons Inc. 2002.

[38] CHANG, Daniel T. y POOLE John D. CWM-Extending UML for Data Warehousing and Business Intelligence, 2000 (en línea). http://xml.coverpages.org/CWM-UMLDWBI.pdf

[39] OMG. CWM Metadata Interchange Patterns Specification, 2004 (en línea). http://www.omg.org/spec/MIPS/1.0/PDF

[40] BROWN, Alan; CONALLEN, Jim y TROPEANO, Dave. Introduction: Models, Modeling, and Model-Driven Architecture (MDA). En: BEYDEDA, Sami, BOOK, Matthias y GRUHN, Volker. Model-Driven Software Development. Berlín: Springer Verlag, 2005. p. 1-16.

[41] DEBNATH, N. y otros. Improving Model Driven Architecture with Requirements Models. IEEE Fifth International Conference on Information Technology: New Generations. 2008.

[42] GREEN, Rebecca; PEARL, Lisa; DOOR, Bonnie y RESNIK, Philip. Lexical resource integration across the syntax semantics interface. Institute for Advanced Computer Studies. Department of Computer Science. College of Information Studies. University of Maryland. 2001.

[43] STAHL, T. y VOLTER, M. Model-Driven Software Development. Technology, Engineering, Management. Willey publications. 2006.

[44] METZGER, Andreas. A systematic look at Model Transformation. En: BEYDEDA, Sami, BOOK, Matthias y GRUHN, Volker. Model-Driven Software Development. Berlín: Springer Verlag, 2005. p. 19-33.

[45] ZHU, Hong y SHAN, Lijun. The CAMLE Modeling Language and Automated Tools. En: BEYDEDA, Sami, BOOK, Matthias y GRUHN, Volker. Model-Driven Software Development. Berlín: Springer Verlag, 2005. p. 57-89.

[46] GRUNSKE, Lars y otros. Using Graph Transformation for Practical Model-Driven Software Engineering. En: BEYDEDA, Sami, BOOK, Matthias y GRUHN, Volker. Model-Driven Software Development. Berlín: Springer Verlag, 2005. p. 91-117.

[47] ATKINSON, Colin y KUHNE, Thomas. A Generalized Notion of Platforms for Model-Driven Development. En: BEYDEDA, Sami, BOOK, Matthias y GRUHN, Volker. Model-Driven Software Development. Berlín: Springer Verlag, 2005. p. 119-136.

[48] JURGENS, Jan. y SHABALIN, Pasha. Tool Support for Model-Driven Development of Security-Critical Systems with UML. En: BEYDEDA, Sami, BOOK, Matthias y GRUHN, Volker. Model-Driven Software Development. Berlín: Springer Verlag, 2005. p. 35-55.

[49] HAMMOUDA, Imed. A Tool Infrastructure for Model-Driven Development Using Aspectual Patterns. En: BEYDEDA, Sami, BOOK, Matthias y GRUHN, Volker. Model-Driven Software Development. Berlín: Springer Verlag, 2005. p. 139-177.

[50] ZHANG, Jing; LIN, Yuehua y GRAY, Jeff. Generic and Domain-Specific Model Refactoring Using a Model Transformation Engine. En: BEYDEDA, Sami, BOOK, Matthias y GRUHN, Volker. Model-Driven Software Development. Berlín: Springer Verlag, 2005. p. 199-217.

[51] EGYED, Alexander. Automatically Discovering Transitive Relationships in Class Diagrams. En: BEYDEDA, Sami, BOOK, Matthias y GRUHN, Volker. Model-Driven Software Development. Berlín: Springer Verlag, 2005. p. 179-198.

[52] LIN, Yuehua; ZHANG, Jing y GRAY, Jeff. A Testing Framework for Model Transformations. En: BEYDEDA, Sami, BOOK, Matthias y GRUHN, Volker. Model-Driven Software Development. Berlín: Springer Verlag, 2005. p. 219-235.

[53] SILAGHI, Raul y STROHMEIER, Alfred. Parallax - An Aspect-Enabled Framework for Plug-in-Based MDA Refinements Towards Middleware. En: BEYDEDA, Sami, BOOK, Matthias y GRUHN, Volker. Model-Driven Software Development. Berlín: Springer Verlag, 2005. p. 237-267.